Digital Image Processing hw2

Bodong Zhang u0949206

4(a) First we need to divide an image into blocks whose size is 32*32. Maybe the width and the height of the image is not multiple of 32. So we get the floor(width/32) and floor(height/32), so they are the maximum number of lines and columns. We scan and process blocks one by one since they are independent with each other.

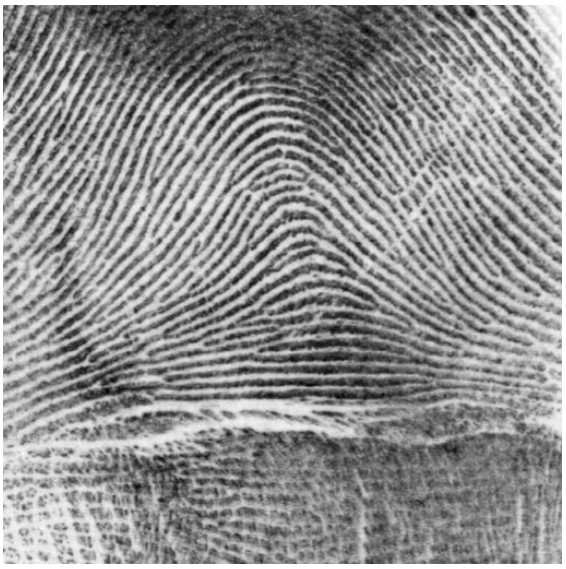
To have the desired mean and variance, we need first calculate the original mean and variance. Assume X is the distribution, then (X-mean(X))/sigma(X) would be the distribution N(0,1). Then we change the N(0,1) distribution to distribution with mean equals 128 and standard deviation equals 50. (N(0,1)*50)+128 would be the desired distribution. In sum, by doing

((X-mean(X))/sigma(X))*50+128

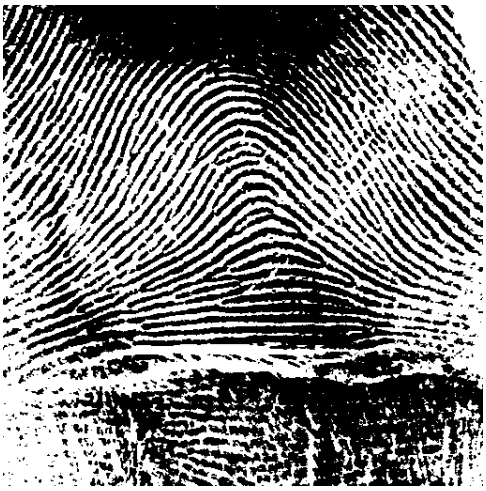
The distribution would be the desire distribution.
The example image is

For blocks with very flat intensity, the intensity values would be stretched largely because the variance need to be fulfilled, which makes the blocks have noise-like intensities.

4(b) The finger print image is

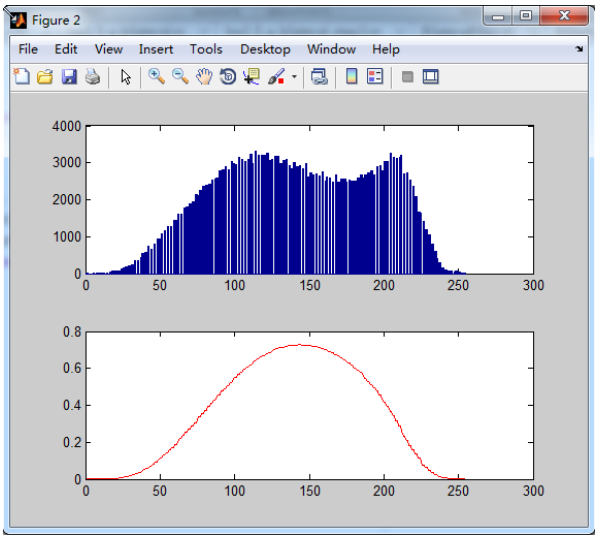


① Just apply Otsu's thresholding



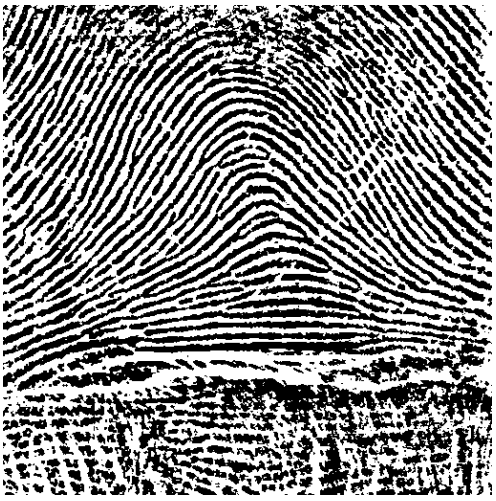
Result after Otsu's thresholding

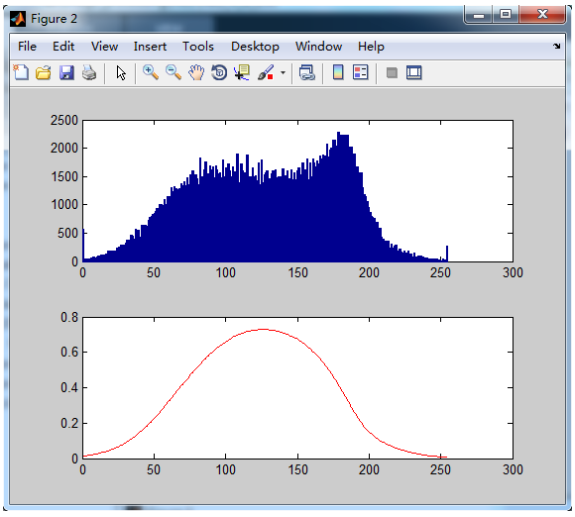
The threshold is 143



histogram before Otsu's thresholding

② First apply AdaptiveEqualize function and then apply Otsu's thresholding.





Result after Otsu's thresholding histogram before Otsu's thresholding
The threshold is 126.
The approach of first applying AdaptiveEqualize function and then applying Otsu's method has more accurate result. The reason is that in some areas the intensity of background is as deep as ridges in other areas. By dividing the image into blocks and transform to same distribution, the background with deep color would be lighter. As a result, the ridges can be precisely detected.

5(a) To do the Gaussian filtering, we need to have integrate the intensity values with weight which fade with the distance to the point(x1, y1) you want to calculate. The formula is

$\Sigma(x2)\Sigma(y2)$ f(x1,y1)*exp(-0.5*(((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/(sigma*sigma)))

which |x2-x1|<3*sigma, |y2-y1|<3*sigma

Then we need to normalize the intensities to preserve the dc component.

It is a little tricky to process the border, my algorithm is that we stop the scan if (x2,y2) has reached the edge even if its distance to (x1,y1) has not reached 3*sigma. By this way, the dc component would still never change after corresponding normalization.

5(b) The formula of Bilateral Filtering is

$H(x1,y1=)k^{-1}(x1,y1)\ \Sigma(x2)\Sigma(y2)\ \{f(x2,y2)*c((x1,y1),(x2,y2))*s(f(x2,y2),f(x1,y1))\}$

$c((x1,y1),(x2,y2))= \exp(-0.5*(((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/(sigmad^2)))$

$s(f(x2,y2),f(x1,y1))= \exp(-0.5*(((f(x2,y2)- f(x1,y1))*(f(x2,y2)- f(x1,y1)))/(sigmar^2)))$

We use the same method to process border. By the algorithm of Bilateral Filtering, we have Gaussian filter to smooth the noise and also we have the range filter to preserve the edge, we only do low-pass filtering in area with similar intensities.

5(c)

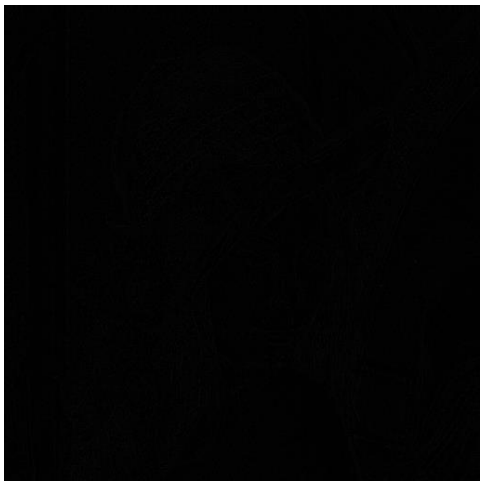
Original image

Gaussian filter

Bilateral filter

The Gaussian filtered image is vague because the details have been smoothed. But for Bilateral filtered image, the edges are preserved

5(d)

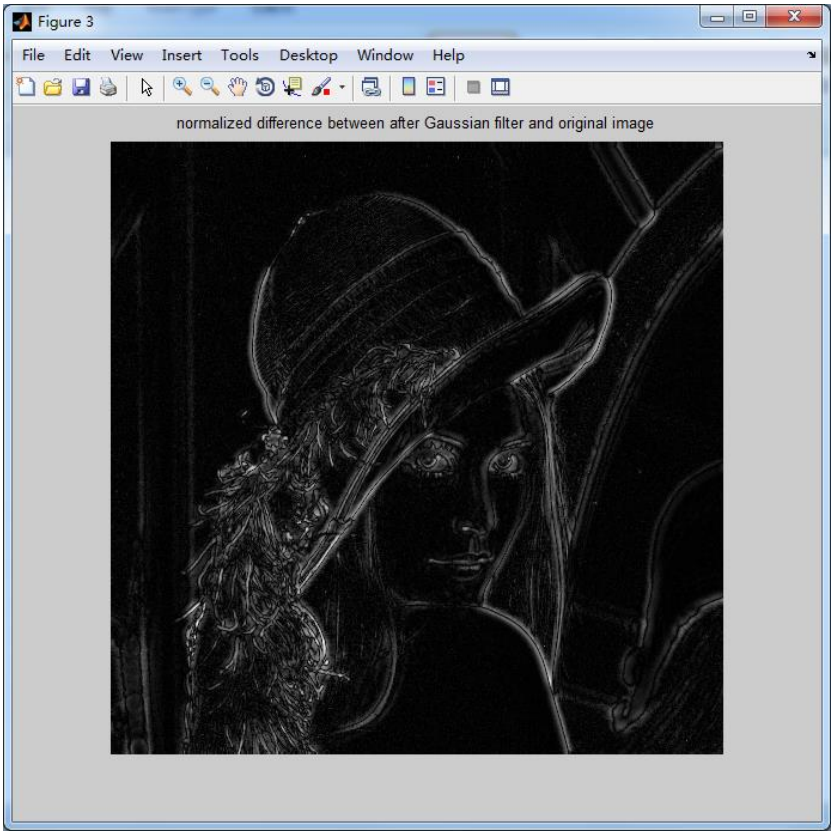


|After Gaussian- original image| |After Bilateral- original image|

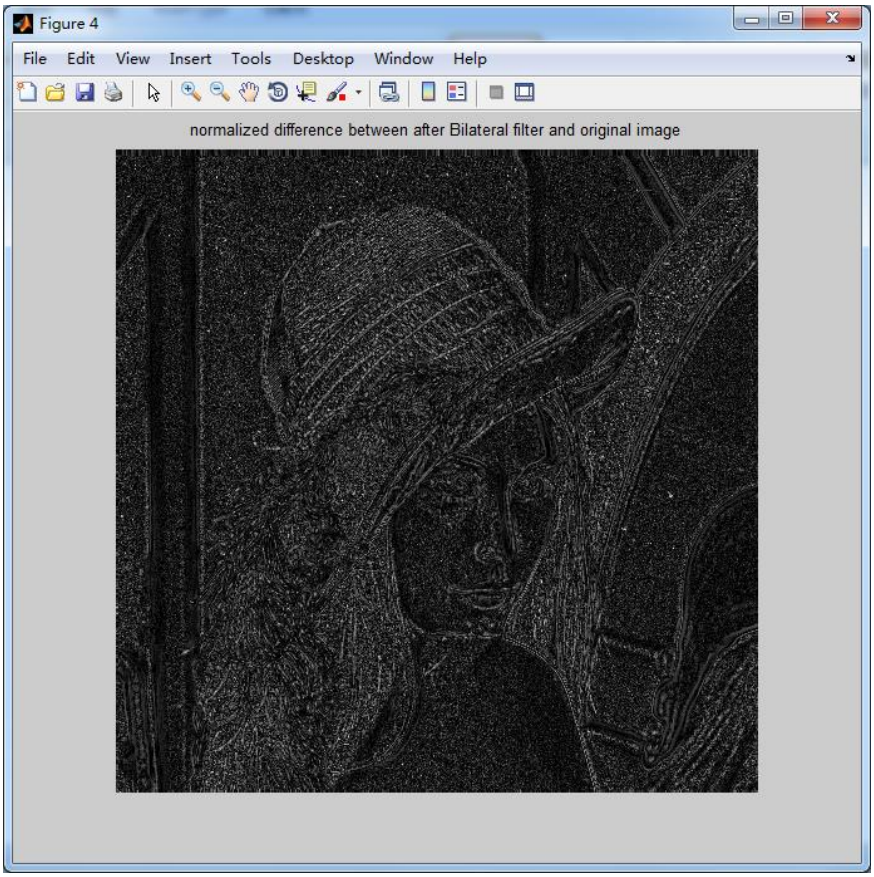
The maximum absolute value of difference with Gaussian filter is 0.4392*255,

The maximum absolute value of difference with Bilateral filter is 0.0627*255,

After normalization, the results are



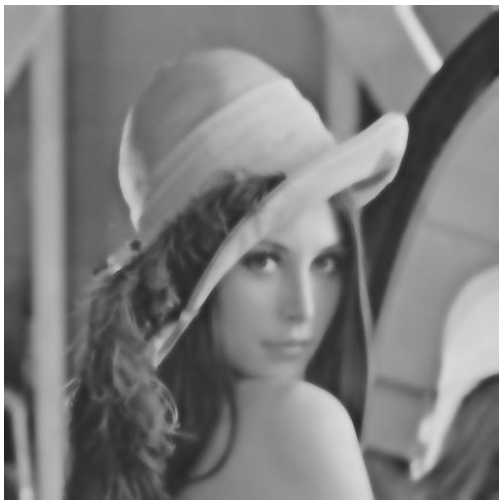
Normalized difference between original image and after Gaussian filter



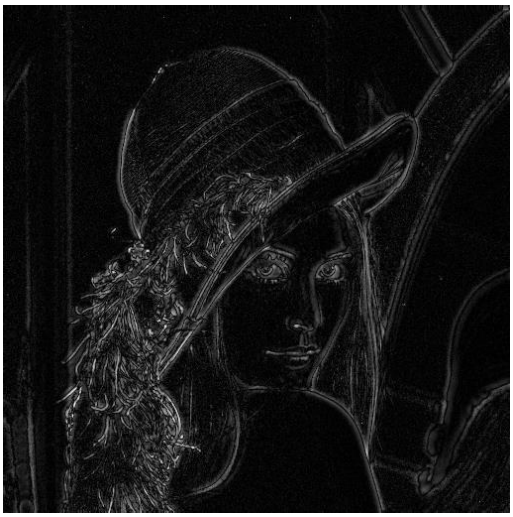
Normalized difference between original image and after Bilateral filter

From the results, we can see that Gaussian filter mainly influences the edges if there is no much noise. But Bilateral filter would not make the edges vague.

5(e)



Bilateral filtered image(sigmad=3,sigmar=100)

Difference to original image Difference to original image(normalized)

The Bilateral filter works like low-pass filter if sigmar=100 because the range filter has less effect since sigmar is large and the exp value for integration tend to be 1 all the time.

5(f) if sigmad→∞, then c function would always be 1, so the bilateral filter is equivalent to range filter.

5(g) if sigmar→∞, then s function would always be 1, so the bilateral filter is equivalent to domain filter(gaussian filter).

6(a) First, we add noise to f

g=f+20*randn(size(f))

After adding noise, if the intensity exceed 255, we would set it to 255, if the intensity is lower than 0, we would set it to 0.

Original image

After Gaussian filtering,


Adding noise After Gaussian filtering

The mean square distance after adding noise is 397.6034

The mean square distance after Gaussian filtering is 214.8330

We see the improvement after Gaussian filtering.

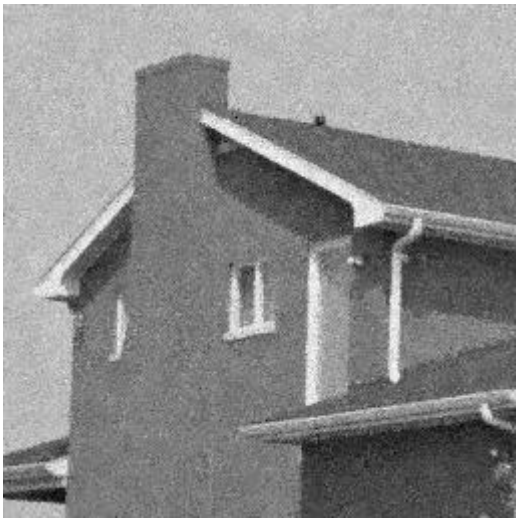
6(b) First, we add noise

g=f+20*randn(size(f))

After adding noise, if the intensity exceed 255, we would set it to 255, if the intensity is lower than 0, we would set is to 0.

After Bilateral filtering,


Sigmar=5(MSE= 369.3725) Sigmar=10(MSE= 296.1979)



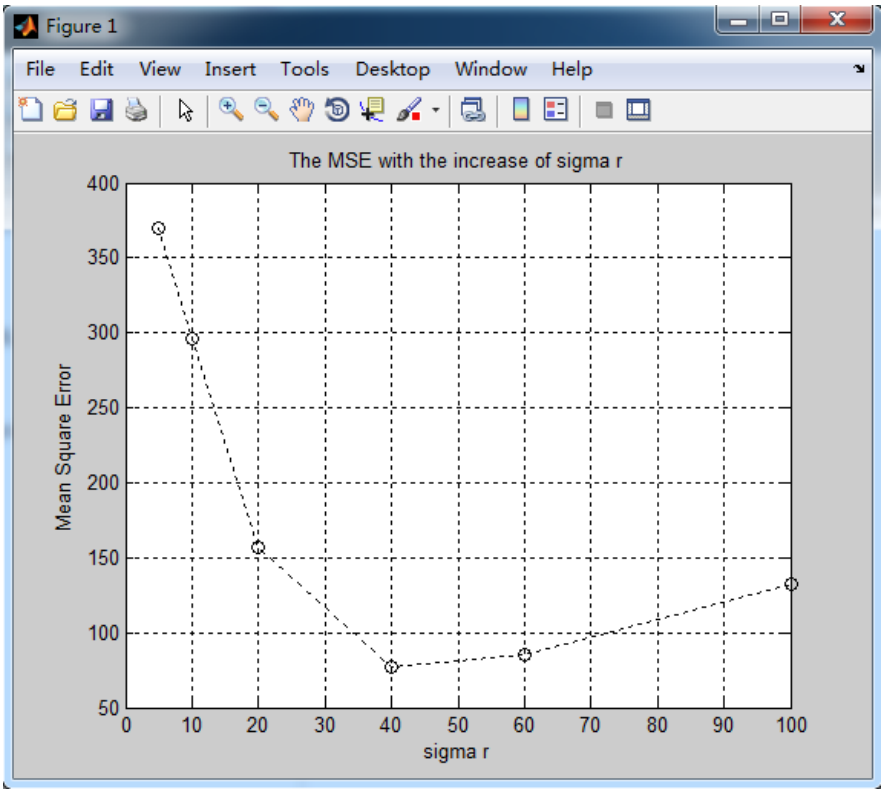
Sigmar=20(MSE= 157.3933)

Sigmar=40(MSE= 76.9235)

Sigmar=60(MSE= 85.7509)

Sigmar=40(MSE= 132.3861)

The MSE with the increase of sigmar works like



At the beginning, if sigmar is too small in comparison with the variation of noise, the noise points would look like sharp edge because the intensity difference would be considered too large. So with the increase of sigmar, the MSE would decrease rapidly since the noise would be smoothed. But if sigmar is too large, then the intensity difference of edge would be considered as noise, so the processed image would be vague, which makes the MSE increase a little bit.